# Teoria dos problemas completos em NP
## algumas notas

2006/07

**Armando B. Matos**

## Índice



**Nota.** Estes apontamentos são apenas um sumário da teoria das classes de complexidade, incluindo em particular o estudo da classe dos problemas completos em NP. O aluno deve também consultar as referências dadas no final, em particular a referência [1].

# Conceitos fundamentais (a estudar)

1. Pincípios básicos
   (a) Problemas de decisão, Codificações e linguagens
   (b) Reduções entre problemas: redução de Karp ("many to one") e de Turing (máquinas com oráculo). Exemplos.
2. As classes P e NP
   (a) As classes P, NP e co-NP
   (b) Problemas completo em NPs: teorema de Cook, outros problemas completo em NPs.
   (c) A hirarquia polinomial.
3. As classes de problemas aleatorizados
   (a) Exemplo: os primos
   (b) Classe BPP
4. A classe IP
   (a) Exemplo: isomorfismo entre 2 grafos.
   (b) Relação com as outras classes.

# Problemas de decisão e linguagens associadas

**Problemas de decisão**: Problemas cuja resposta é SIM ou NÃO

**Exemplos.** SAT (instância $(U, C)$), 3SAT, 2-COL, 3-COL, Clique, Partição, etc.

**Codificação:** Função $c : \mathcal{I} \to \Sigma^\star$ do conjunto das instâncias nas palavras de um alfabeto.

**Exemplo.** Duas possíveis representações do mesmo grafo:
- "`3,(1,1),(1,3),(2,3)`", alfabeto `{0,1,,,,,9,'(',')',',',}`,
Representação: número de vértices, lista de ramos.
- `101001000`, alfabeto `{0,1}`
Representação: matriz de adjacências. **Exercício.** Desenhe o grafo correspondente.

**Linguagem associada a um problema $\pi$ com uma codificação $c$:**

$$L_\pi^c = \{c(I) \mid \text{A resposta para a instância } I \text{ é SIM}\}$$

Note-se que $L_\pi^c \subseteq \Sigma^\star$. Em geral ignora-se a codificação e fala-se simplesmente na linguagem associada a um problema de decisão.

**Exemplo.** "$(3, (4, (1, 3), (2, 3), (1, 4), (3, 1)))$" (que tem a forma $(n, \text{grafo})$) não pertence a $L_{\text{Clique}}$.

**Nota.** Convencionamos que as palavras sintacticamente incorrectas não pertencem à linguagem.

# Redução polinomial entre problemas

**Redução polinomial entre problemas**[1]
Diz-se que o problema $A$ se reduz polinomialmente ao problema $B$ e escreve-se $A \propto B$ se existir uma função $f : \Sigma_A^\star \to \Sigma_B^\star$ (onde $\Sigma_A$ e $\Sigma_B$ são os algabetos, $A \subseteq \Sigma_A^\star$, $B \subseteq \Sigma_B^\star$) tal que

1. $x \in A$ sse $f(x) \in B$.
2. $f(x)$ é computável em tempo polinomial em $|x|$.

**Nota.** Escreve-se também $A \leq_p B$ em vez de $A \propto B$.

**Exemplo.** 3SAT$\propto$VC, vamos mostrar isso!
Ver também a referência [1], págs. 54–56.

**Os problemas:**
**Problema 3SAT** (Satisfabilidade de claúsulas com 3 literais)
**Instância.** $(U, C)$ onde $U$ é um conjunto de variáveis lógicas e $C$ é um conjunto de claúsulas (CNF) de 3 literais. Cada literal é uma variável de $U$ ou uma variável de $U$ negada.
**Pergunta.** Existe alguma valorização lógica das variáveis de $U$ que satisfaça todas as claúsulas?

**Problema VC** (Cobertura por vértices)
**Instância.** Um grafo não dirigido $G = (V, E)$ e um inteiro $k \leq |V|$.
**Pergunta.** Existe algum conjunto $V' \subseteq V$ com não mais de $k$ vértices ($|V'| \leq k$) tal que qualquer ramo tem pelo menos uma das extremidades em $V'$?

[1] Este conceito só difere do conceito de redução entre problemas de decisão que foi tratado na parte da matéria "Computabilidade para Complexidade" pelo facto de se exigir que a função de transformação seja computável em tempo polinomial em $|x|$.

**Pincípios básicos**
**Exemplo de redução polinomial**

**A redução 3SAT$\propto$VC**

Objectivo. Transformar cada instância $(U, C)$ de SAT numa instância $(k, G) = f(U, C)$ de VC

$$(U, C) \xrightarrow{f} (k, G)$$

de modo que (i) $(U, C)$ tem solução no problema 3SAT sse $(k, G)$ tiver solução no problema VC. (ii) A transformação é efectuada em tempo polinomial.

Transformação de $(U, C)$ em $(k$, Grafo).

1. O grafo $G$:
   (a) Para cada $u_i \in U$ criamos 2 vértices $u_i$ e $u'_i$ e o ramo $(u_i, u'_i)$ ($u_i$ representa no grafo a variável lógica e $u'_i$ a variável negada).
   (b) Para cada claúsula $(l_1, l_2, l_3)$ de $C$ (cada $l_i$ é um literal) criamos 3 vértices $l_1$, $l_2$ e $l_3$ bem como o triângulo $(l_1, l_2)$, $(l_2, l_3)$, $(l_3, l_1)$.
   (c) Cada vértice de cada triângulo é ligado liga à variável correspondente; isto é, se o literal é da forma $u_i$, criamos um ramo entre esse vértice do triângulo e o vértice $u_i$, e se o literal é da forma $\overline{u_i}$, criamos um ramo entre esse vértice do triângulo e o vértice $\overline{u_i}$.
2. O valor de $k$ é $|U| + 2|C|$

Note que o grafo tem $2|U| + 3|C|$ vértices e $2|U| + 6|C|$ ramos.

**Teorema 1** *Seja $G = (V, E)$ o grafo definido atrás e $k = |U| + 2|C|$.*
*- Qualquer cobertura por vértices de $G$ tem pelo menos $k$ vértices.*
*- Existe uma cobertura de $k$ vértices de $G$ se e só a instância do problema original 3-SAT for satisfazível.*
*- A construção do grafo pode ser efectuada em tempo polinomial em $|U| + |C|$.*

**Exercício.** Demonstre!

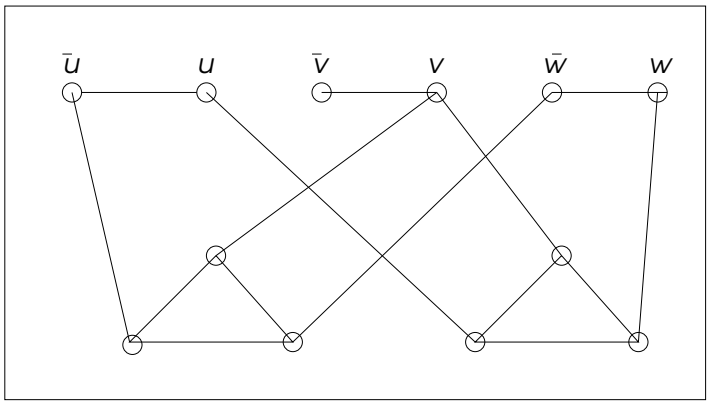



Figura 1: Exemplo da transformação associada à redução 3SAT$\propto$VC.

**Exemplo.** A instância de SAT

$$U = \{u, v, w\}, \quad C = \{(\overline{u}, v, \overline{w}), (u, v, w)\}$$

tem como transformada $(G, 7)$ em que $G$ é o grafo representado na Figura 1.

# Alguns problemas de decisão importantes

É necessário conhecê-los!

**Problema 3SAT** (Satisfabilidade de claúsulas com 3 literais)
Já definido.

**Problema SAT** (Satisfabilidade de claúsulas)
Definido de forma análoga a 3SAT, mas sem restrições no número de literais por claúsula (podem existir em $C$ claúsulas com um qualquer número de literais).

**Problemas 1SAT, 2SAT, etc.**
Definido de forma análoga a 3SAT.

**Problema TAUT** (Tautologia)
**Instância.** Uma expressão proposicional $E$ que pode conter os quantificadores $\neg$ (negação), $\wedge$ (conjunção) e $\vee$ (disjunção), bem como variáveis lógicas.
**Pergunta.** A expressão $E$ assume o valor VERDADE quaisquer que sejam os valores lógicos atribuídos às variáveis lógicas?

**Problema CLIQUE**
**Instância.** Um grafo não dirigido $G = (V, E)$ e um inteiro $k \leq |V|$.
**Pergunta.** Existe algum conjunto $V' \subseteq V$ com $k$ vértices tal que $(u, v) \in E$ para quaisquer vértices distintos $u$ e $v$ de $V'$?

**Problema VC** (Cobertura por vértices)
Já definido.

**Problema HC** (ciclo hamiltoniano).
**Instância.** Um grafo não dirigido $G$.
**Pergunta.** Existe em $G$ um ciclo que passe uma e uma só vez por cada vértice do grafo?

**Problema HP** (Caminho hamiltoniano).
**Instância.** Um grafo não dirigido $G$.
**Pergunta.** Existe em $G$ um caminho que passe uma e uma só vez por cada vértice do grafo? Nada se impõe sobre os vértices extremos do caminho, para além de serem distintos.

**Problema PART** (Partição)
**Instância.** Um multiconjunto de inteiros positivos $A = \{a_1, a_2, \cdots, a_n\}$
**Pergunta.** Existe uma partição de $A$ em dois subconjuntos $A_1$ e $A_2$ tal que $\Sigma_{u \in A_1} u = \Sigma_{u \in A_1} u$?

**Problema 3DM** ("3-dimensional matching")
**Instância.** Conjuntos disjuntos $A$, $B$ e $C$, todos com o mesmo número $m$ de elementos e um conjunto de trios $Z \subseteq A \times B \times C$.
**Pergunta.** Existem $m$ trios em $Z$, $(a_1, b_1, c_1), \ldots, (a_m, b_m, c_m)$ tais que $A = \cup_{i=1}^{m} a_i$, $B = \cup_{i=1}^{m} b_i$ e $C = \cup_{i=1}^{m} c_i$ (isto é, tal que os primeiros elementos dos trios são todos distintos, o mesmo acontecendo com os segundos e terceiros elementos)?

**Problemas 2DM, 4DM, etc.**
Definidos de forma análoga.

**Outros exemplos de redução polinomial — Exercícios!**

Em cada caso deve definir a função de transformação, mostrar que o problema transformado tem solução se e só se (é essencial considerar os 2 sentidos!) o problema original tiver solução e argumentar que a transformação é implementável através de um algoritmo que corre em tempo polinomial. Este último aspecto, embora tão importante como os outros, não requer uma demonstração detalhada que seria, na maior parte dos casos, fastidiosa e sem interessa.

[05] **Exercício.** Mostre que 3SAT$\propto$SAT.

[25] **Exercício.** Mostre que SAT$\propto$3SAT.

[15] **Exercício.** Mostre que VC$\propto$CLIQUE.

[15] **Exercício.** Mostre que IV$\propto$CLIQUE, onde IV é o problema dos vértices independentes: dado um par $(G, k)$ onde $G$ é um grafo não dirigido, $G = (V, E)$ e o inteiro $k \leq |V|$, pergunta-se: existe um subconjunto $U' \subseteq U$ de não mais que $k$ vértices todos "desligados" entre si (isto é tais que se $u$ e $v$ são vértices distintos de $U'$ então $(u, v) \notin E$)?

[15] **Exercício.** Mostre que HC$\propto$HP onde HC é o problema do ciclo hamiltoniano e HP é o problema do caminho hamiltoniano.

[15] **Exercício.** Mostre que HP$\propto$HC.

[20] **Exercício.** Mostre que se existir um algoritmo polinomial para o problema de decisão CLIQUE, também existe

1. Um algoritmo polinomial para o problema de determinar o tamanho do maior "clique" de um grafo dado.
2. Um algoritmo polinomial para o problema de determinar um "clique" de tamanho máximo de um grafo dado.

[30] **Exercício.** Mostre que 3SAT$\propto$3COL, onde 3COL é o problema seguinte.
**Instância.** Um grafo não dirigido $G = (V, E)$.
**Pergunta.** $G$ pode ser colorido com 3 cores, isto é, existe uma função $f : V \to \{1, 2, 3\}$ tal que $(u, v) \in E \Rightarrow f(u) \neq f(v)$?

# Classes P, NP, co-NP

**Classes P e NP**

**Classe P (Polinomial)**
Informalmente: problemas de decisão - ou, de forma equivalente, linguagens - que podem ser resolvidos em tempo polinomial.

**Definição** Classe das linguagens $L$ para as quais existe um algoritmo polinomial em $|x|$ que calcula o predicado $r_L(x) =$VERDADE se $x \in L$, FALSO se $x \notin L$.
De outro modo: é a classe das linguagens

$$\cup_{i \geq 0} \Pi(n^i)$$

onde $\Pi(n^i)$ designa a classe das linguagens *para as quais existe* um algoritmo polinomial de decisão. Note que o importante é existirem ou não algoritmos polinomiais de decisão.

**Exemplos.** 1-COL, 2-COL, 1-SAT, 2-SAT, qualquer linguagem regular.

**Classe NP (Não determinístico, polinomial)**
Informalmente: problemas de decisão que têm uma solução verificável em tempo polinomial.

**Definição** Classe das linguagens $L$ para as quais existe um polinómio $p()$ e predicado $r(x, y)$ tais que

$$x \in L \text{ sse } \exists y \text{ com } |y| \leq p(|x|) \text{ tal que } r(x, y)$$

sendo $r()$ computável em tempo polinomial em $|x|$ (ou de forma equivalente em $|x| + |y|$).

**Exemplos.** 1-COL, 2-COL, COL, 1-SAT, 2-SAT, SAT, qualquer linguagem pertencente a P.

**Definição** Seja $\mathcal{R}$ uma classe de linguagens. A classe co-$\mathcal{R}$ é a classe de linguagens que são complementares das linguagens de $\mathcal{R}$.
Nota No caso das linguagens derivadas dos problemas de decisão estamos a ignorar as palavras sintacticamente incorrectas, isto é, que não representam nenhuma instância. Por outras palavras, o universo relativamente ao qual se faz a complementação é constituído apenas pelas palavras sinctaticamente correctas de $\Sigma^\star$ (e não de $\Sigma^\star$).

**Teorema** P = co-P.

**Classe co-NP**
Informalmente: problemas de decisão tais que, o eventual facto de não existir solução é verificável em tempo polinomial.

**Definição** A classe NP é a classe das linguagens $L$ para as quais existe um polinómio $p()$ e predicado $r(x, y)$ tais que $x \in L$ sse $\forall y$ com $|y| \leq p(|x|)$ é $r(x, y)$ sendo $r()$ computável em tempo polinomial em $|x|$.

**Exemplos.** TAUT (uma expressão lógica é uma tautologia?). COMPOSTO (o número dado é composto?). Qualquer linguagem complementar de uma linguagem de NP. Qualquer linguagem de P.

**Teorema** P $\subseteq$ NP $\cap$ co-NP

# Problemas completos e teorema de Cook

**Definição** Um problema $\Pi$ dix-se *completo* numa certa classe $\mathcal{R}$ se $\Pi$ pertence a $\mathcal{R}$ e qualquer problema de $\mathcal{R}$ reduz-se polinomialmente a $A$. Ou seja

$$\Pi \in \mathcal{R} \ \wedge \ \forall A \in R \ A \propto \Pi$$

**Nota.** Em vez de "completo em $\mathcal{R}$" também se diz "$\mathcal{R}$–completo".

Os problemas completos de uma classe são os problemas mais difíceis dessa classe: a sua eventual solução polinomial forneceria a solução polinomial para todos os problemas dessa classe.

A existência de problemas completos - bem como a possibilidade de demonstrar esse facto - é espantosa (de a classe não estiver contida em P) pois implica a existência de uma infinidade de reduções polinomiais!

**Teorema de Cook**. O problema SAT é um problema completo em NP (também se diz "NP–completo" ou "NPC").

A ideia da demonstração é a seguinte: se $\Pi$ é um problema NP, existe uma máquina de Turing não determinística $M$ e um polinómio $p()$ tais que $x \in \Pi$ sse existe uma computação aceitadora de $x$ com não mais de $p(|x|)$ passos. Este facto é traduzido num conjunto de claúsulas (com tamanho polinomial)!

## Teorema de Cook - ideia da demonstração

Construiu-se um esquema de reduções polinomiais que funciona para qualquer linguagem $L \in$ NP. A construção é sempre efectuada em tempo polinomial.
$x \in L$ se e só se...

...Existe uma computação de $M$ que aceita $x$ se e só se...

...Existe uma computação de $M$ que aceita $x$ com $\leq p(n)$ transições na fase de verificação e com um $y$ nos índices $-p(n)$,...,$-1$ se e só se...

...O conjunto das claúsulas $C_L$ é satisfazível.

**Variáveis lógicas**:

| Variáveis | Significado |
|---|---|
| $Q[i,k]$ | No instante $i$ $M$ está no estado $k$ |
| $H[i,j]$ | No instante $i$ a cabeça está na célula n. $j$ |
| $S[i,j,m]$ | No instante $i$ a célula $j$ tem o símbolo $m$ |

Onde $0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)$, $0 \leq k \leq r$ e $0 \leq m \leq s$, sendo $n$, $r$ e $s$ respectivamente o comprimento de $x$, o número de estados e o tamanho do alfabeto da fita.

**Claúsulas**:

- Grupo I: Em cada instante a máquina está num e num só estado. Claúsulas (em número de $(p(n)+1)(1+(r+1)(r/2))$):
  $(Q[i,0], Q[i,1], \cdots, Q[i,r])$ para $0 \leq i \leq p(n)$
  $(Q'[i,j_1], Q'[i,j_2])$ para $0 \leq i \leq p(n)$, e $0 \leq j_1 < j_2 \leq r$
- Grupo II: Em cada instante a cabeça está sobre uma e uma só célula.
- Grupo III: Em cada instante cada célula contém um e um só símbolo.
- Grupo IV: No instante 0 a máquina está na configuração inicial.
  $(Q[0,0])$, $(H[0,1])$, $(S[0,0,0])$
  $(S[0,1,a_1])$, $(S[0,2,a_2]), \cdots, (S[0,n,a_n])$
  $(S[0,n+1,0])$, $(S[0,n+2,0]), \cdots, (S[0,p(n)+1,0])$
  onde $x = a_1 a_a \cdots a_n$.
- Grupo V: No instante $p(n)$ a máquina está (ou melhor, pode estar) no estado final $y$ tendo por isso aceite $x$. $(Q[p(n),1])$
- Grupo VI: Para cada instante $i$, $0 \leq i < p(n)$, a configuração no instante $i+1$ é obtida pela aplicação da função de transição $\delta$.

Note-se que não impusemos o conteúdo inicial das células de índice negativo. Esse facto permite tratar de uma forma fácil e não-determinismo.

# Problemas completos em NP

Se $\Pi$ for um problema de NP e mostrarmos que SAT $\propto \Pi$ (ou que outro problema que já sabemos ser completo em NP se reduz polinomialmente a $\Pi$), fica provado que $\Pi$ também é completo em NP. Assim se mostraram que muitos problemas são completo em NPs como, por exemplo:

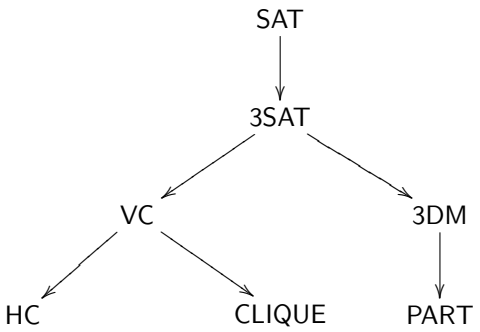
- 3SAT, 4SAT, etc
- 3COL, 4COL, etc
- Clique, VC
- Partição
- Determinar se 2 grafos dados são isomorfos

**Problemas completos em NP**

Mostrou Cook: $\forall \Pi \in$ NP : $\Pi \propto$SAT
Nós mostramos (exercício): SAT$\propto$3SAT
Logo, como 3SAT é NP, 3SAT é também completo em NP.
Também se prova ou já provamos

3SAT$\propto$VC | 3SAT$\propto$3DM | VC$\propto$HC | VC$\propto$CLIQUE | 3DM$\propto$PART

ou, representado em grafo:




Por transitividade vem para todos os problemas $X$ na árvore (X=3SAT, X=VC, etc.)

$$\forall \Pi \in \text{NP} : \Pi \propto X$$

*Logo, todos estes problemas são também completos em NP!*

# Mostrando que um problema é completo em NP

Suponhamos que já sabemos que o problema $\Pi$ pertence a NP. Coo mostrar que $\Pi$ é completo em NP?

**2 modos**

1. "À la Cook", isto é, partindo da definição de completo. Teríamos de mostrar que todos os problemas de NP se podem reduzir polinolialmente a $\Pi$, ver a demonstração do Teorema de Cook.

2. Usando um problema A que já sabemos ser completo em NP, mostramos que

   $$\text{A} \propto \Pi$$

   Por transitividade da relação $\propto$, de $\forall X \in \text{P}\ X \propto \text{A}$ e de $\text{A} \propto \Pi$ concluimos que $\forall X \in \text{P}\ X \propto \Pi$, isto é, que $\Pi$ é completo em NP.

O método 1. é directo mas muito trabalhoso; o método 2. é usualmente muito mais simples e é o que é usado na prática.

---

**Nota importante.** Como problema A escolhido para se mostrar que A$\propto \Pi$, poderíamos escolher qualquer problema que já sabemos ser completo em NP. Na prática escolhe-se um dos seguintes seis problemas

Os 6 problemas básicos

3SAT | CLIQUE | VC | HC | PART | 3DM

**Porquê?** (i) simplicidade (por exemplo, 3SAT mas não SAT nem 4SAT; 2SAT é P) que ajuda à demonstração, (ii) demonstrações de completude em NP já bem confirmadas.

**É sempre um destes problemas que o aluno deve escolher nos exercícios!**

## Mostrando que um problema é completo em NP – exercício

**Exercício.** Mostre que o problema 4SAT é completo em NP.
**Resolução.** Primeiro mostramos que 4SAT$\in$ NP. Seja $(U, C)$ uma instância de 4SAT. Se nos for dada uma valorização lógica das variáveis de $U$ (isto é, uma possível "solução"), podemos determinar em tempo polinomial o valor lógico da conjunção das claúsulas em $C$.

Agora, escolhemos 3SAT como problema completo em NP e vamos (tentar...) mostrar que

$$\text{3SAT} \propto \text{4SAT}$$

Quando mostrarmos este facto fica acabado o exercício!

Note-se que a demonstração de redução 3SAT$\propto$4SAT não ajudaria ao fim em vista (**Exercício.** em que ajudaria?).

Temos que transformar cada instância $(U, C)$ de 3SAT numa instância $(U', C')$ de 4SAT de modo que $(U, C)$ *tem solução em 3SAT sse* $(U', C')$ *tiver solução em 4SAT*.
Comecemos com um exemplo que ilustra o caso geral. Seja

$$C = \{(l_1, l_2, l_3),\ (l_4, l_5, l_6)\}$$

e sejam $u$ e $v$ novas variáveis lógicas, isto é, que não ocorrem em $U$.

**(continução)**

A transformada é $(U', C') = f(U, V)$:

$$\begin{cases} U' = U \cup \{u, v\} \\ C' = \{(l_1, l_2, l_3, u), (l_1, l_2, l_3, \overline{u}), (l_4, l_5, l_6, v), (l_4, l_5, l_6, \overline{v})\} \end{cases}$$

Esta transformação é implementável em tempo polinomial em $|(U, V)|$ (aliás mesmo em tempo linear).

*Num sentido – a existência de solução do problema $(U, C)$ de 3SAT implica a existência de solução do problema $(U', C') = f(U, C)$ de 4SAT.* Suponhamos que existe uma valorização que satisfaz $C$. Consideremos essa valorização estendida de forma arbitrária a $U'$. Independentemente do valor lógico de $u$, como a claúsula $(l_1, l_2, l_3)$ está satisfeita em $C$, as 2 claúsulas

$$(l_1, l_2, l_3, u),\ (l_1, l_2, l_3, \overline{u}),$$

ficam satisfeitas uma vez que um dos literais (pelo menos) $l_1$, $l_2$ ou $l_3$ tem que ter o valor VERDADE (porque $u$ ou $\overline{u}$ têm o valor FALSO). De forma análoga se conclui que as claúsulas $(l_1, l_2, l_3, u)$ e $(l_1, l_2, l_3, \overline{u})$ estão ambas satisfeitas.

*No outro sentido – a existência de solução do problema $(U', C') = f(U, C)$ de 4SAT implica a existência de solução do problema $(U, C)$ de 3SAT.*
**Nota.** Note-se que apenas interessam problemas de 4SAT que são imagens de problemas de 3SAT.
Suponhamos que existe uma valorização que satisfaz $C'$. Consideremos em particular as claúsulas

$$(l_1, l_2, l_3, u),\ (l_1, l_2, l_3, \overline{u}),$$

Suponhamos primeiro que nessa valorização $u$ tem o valor VERDADE; então $\overline{u}$ tem o valor FALSO pelo que, olhando para a segunda claúsula, $l_1$ ou $l_2$ ou $l_3$ têm que assumir o valor VERDADE. Se $u$ tem o valor FALSO, o raciocínio é idêntico pelo que a claúsula $(l_1, l_2, l_3)$ assume necessariamente o valor VERDADE.
Se considerarmos agora as claúsulas

$$(l_4, l_5, l_6, v),\ (l_4, l_5, l_6, \overline{v}),$$

concluimos assim que, com essa mesma valorização restrita a $U$, a claúsula $(l_4, l_5, l_6)$ tem o valor VERDADE e que, por isso, a instância de 3SAT é satisfazível.

*A generalização a um número qualquer de claúsulas de 3SAT fica a cargo do leitor, não é difícil.*

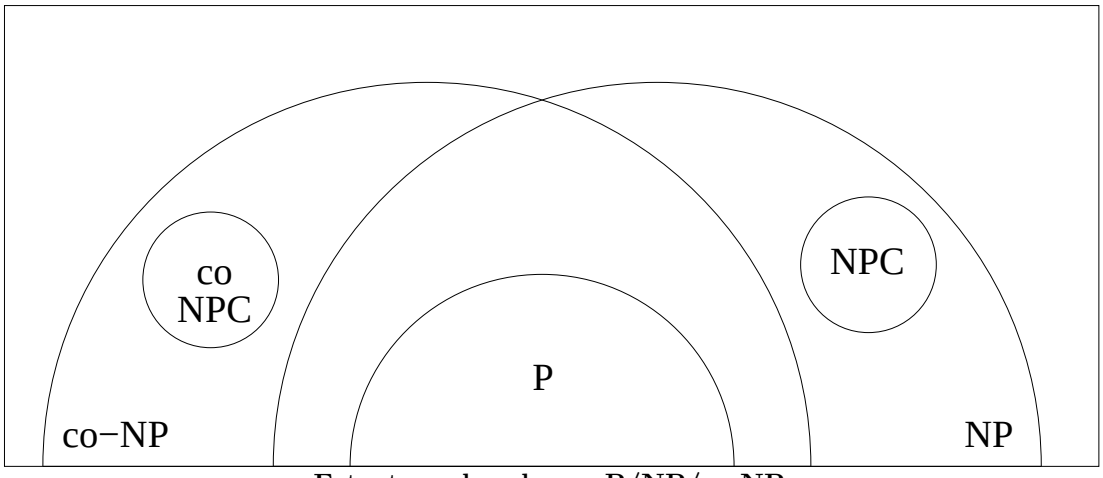
# O significado de um problema ser completo em NP

Acabamos de mostrar que um problema de decisão $\Pi$ é completo em NP (ou, como também se diz "NP-completo").
Quais as implicações? O que é que isso significa na prática?

- *O significado:* Não existem algoritmos que decidam $\Pi$ em tempo polinomial.
  **Notas.** Trata-se de um facto em que se assume ser P$\neq$NP e que é
  - Assintótico.
  - Baseado na análise no pior caso.
- *Problemas de optimização:* Também não existe algoritmo polinomial para o problema de optimização correspondente, se existir. Por exemplo, dado CLIQUE ser completo em NP, também podemos concluir que o problema "dado um grafo determinar o tamanho de um clique máximo" também não tem solução polinomial (Porquê?).
- *Mas, na prática...*
  - As instâncias podem não ser muito grandes de modo que seja viável usar algoritmos exponenciais.
  - Embora qualquer algoritmo possível seja necessariamente exponencial (análise no pior caso), pode não o ser no caso médio ou nas distribuições que ocorrem na prática.

$\Pi$ é completo em NP: um resultado muito forte e com consequências práticas (por exemplo, evita a procura de algoritmos polinomiais), mas não necessariamente dramático.

# Estrutura das classes P/NP/co-NP




**Estrutura das classes P/NP/co-NP.**

# A Hierarquia polinomial

**Definição** Para qualquer $k \geq 1$ uma linguagem $L \in \Sigma^\star$ pertence à classe $\Sigma_k^p$ sse existirem polinómios $p_1, \ldots, p_k$ e um predicado $r$ com $k+1$ argumentos tal que

$$\begin{aligned} x \in L \quad sse \quad & (\exists y_1 \text{ com } |y_1| \leq p_1(|x|)) \\ & (\forall y_2 \text{ com } |y_2| \leq p_2(|x|)) \\ & \cdots \\ & (Q y_k \text{ com } |y_k| \leq p_k(|x|)) \\ & r(x, y_1, \cdots, y_k) \end{aligned}$$

onde o quantificador "$Q$" é "$\exists$" se $k$ é ímpar e "$\forall$" se $k$ é par.
Nota O "$p$" em $\Sigma_k^p$ designa apenas que se trata da hierarquia polinomial evitando assim confusões com um conceito análogo da teoria da computabilidade, a hierarquia aritmética.
Note-se em particular que NP=$\Sigma_1^p$.

**Definição** A *hierarquia polinomial* é formada pelas seguintes classes de linguagens

- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathsf{P}$
- Classe $\Sigma_k^p$ para $k \geq 1$
- Classe $\Pi_k^p = \text{co-}\Sigma_k^p$ para $k \geq 1$.
- Classe $\Delta_k^p$ constituída pelas linguagens reconhecíveis em tempo polinomial por uma máquina com um oráculo para alguma linguagem de $\Sigma_{k-1}^p$.

Em particular

- $\Sigma_1^p = \mathsf{NP}$
- $\Pi_1^p = \text{co-NP}$

**A Hierarquia polinomial - definição equivalente**:
**Definição** Seja $\mathcal{A}$ uma classe de linguagens. Define-se $\mathsf{P}^{\mathcal{A}}$ como sendo a classe de linguagens reconhecíveis em tempo polinomial por máquinas de Turing determinísticas com oráculos para linguagens da classe $\mathcal{A}$.

**Definição** Seja $\mathcal{A}$ uma classe de linguagens. Define-se $\mathsf{NP}^{\mathcal{A}}$ como sendo a classe de linguagens reconhecíveis em tempo polinomial por máquinas de Turing não determinísticas com oráculos para linguagens da classe $\mathcal{A}$.
Analogamente se definem as classes $\mathsf{P}^\pi$ e $\mathsf{NP}^\pi$ onde $\pi$ é um problema fixo.

**Exemplos.**

$$\begin{aligned} &\mathsf{P}^{\mathsf{P}} = \mathsf{P} \\ &\mathsf{NP}^{\mathsf{P}} = \mathsf{NP} \\ &\mathsf{NP} \cup \text{co-NP} \subseteq \mathsf{P}^{\mathsf{NP}} \\ &\mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{NP}^{\mathsf{NP}} \end{aligned}$$

**A Hierarquia polinomial - outra Definição**

**Definição** A hierarquia polinomial é formada pelas classes $\Delta_i^p$, $\Sigma_i^p$ e $\Pi_i^p$ com $i \geq 0$ assim definidas

- $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathsf{P}$
- $\Sigma_i^p = \mathsf{NP}^{\Sigma_{i-1}^p}$
- $\Pi_i^p = \text{co-}\Sigma_i^p$ para $i \geq 1$
- $\Delta_i^p = \mathsf{P}^{\Sigma_{i-1}^p}$ para $i \geq 1$

**A Hierarquia polinomial - Um problema em $\Sigma_2^p$:**

**Problema** EME, Expressão lógica mínima e equivalente.
**Instância.** Uma expressão lógica $E$ podendo envolver variáveis lógicas e conectivos lógicos $\vee$, $\wedge$ e $\neg$ e um inteiro não negativo $k$.
**Pergunta.** Existe uma expressão lógica equivalente a $E$ (isto é que toma o mesmo valor lógico que $E$ qualquer que seja a valorização lógica) de comprimento não inferior a $k$?
Nota: Comprimento de uma expressão lógica: número de ocorrências de variáveis lógicas.
Aparentemente EME não é NP nem co-NP (porquê?). Contudo é fácil ver que é $\mathsf{NP}^{\text{co-NP}}$, utilizando para oráculo o problema da equivalência de expressões lógicas (EL). Pertence também a $\mathsf{NP}^{\mathcal{NP}}$.
Um algoritmo que utiliza EL é o seguinte.

1. Gera não determnisticamente a expressão lógica $E'$ de comprimento $\leq k$.
2. Consulta o oráculo de co-EL tendo como dados $(E, E')$.
3. Se a resposta é NÃO ($E$ e $E'$ são equivalentes), responde SIM e vice-versa.

O problema EME pertence a $\Sigma_2^p$:

$$\exists E',\ |E'| \leq k,\ \forall \mathcal{V}\ \ \underbrace{\mathcal{V}(E) = \mathcal{V}(E')}_{polinomial}$$

onde $\mathcal{V}$ representa a valorização lógica.

**Reduções de Turing**

**Definição** Diz-se que o problema $A$ se reduz por Turing ao problema $B$ e escreve-se $A \propto_t B$ quando $A \in \mathsf{P}^B$. Assim, um eventual algoritmo polinomial para $B$ permitiria obter um algoritmo polinomial para $A$.

Note-se que é possível utilizar o oráculo mais que uma vez (mais concretamente um número polinomial de vezes) e que as suas respostas podem ser usadas da forma que se entender no algoritmo polinomial.

**Teorema** Se $A \propto B$ então $A \propto_t B$.

**Dem** Exercício.

**Teorema** Se NP $\neq$ co-NP então pode acontecer $A \propto_t B$ mas não $A \propto B$.

**Dem** (Ideia) Se $B$ é NP-completo e $A$ é um problema qualquer de co-NP então é $A \propto_t B$. Vamos supor que era também $A \propto B$. Então $A$ passava a ter também testemunhos polinomiais de solução.

**O que se sabe e não sabe**

This is the sad present of Computer Science.

Levin, *Fundaments of Computing*

- Pensa-se que P≠NP mas não existe certeza. Qualquer algoritmo polinomial para qualquer problema $\mathcal{NP}$-completo teria como consequência P = NP. Como há muitos investigadores à procura de bons algoritmos para diversos problemas completo em NPs...
- De modo semelhante pensa-se que NP≠co-NP mas não existe certeza. É claro que se for P=NP, também é NP=co-NP.
- Se para um certo $i \geq 0$ for $\Sigma_i^p = \Sigma_{i+1}^p$ a hierarquia polinomial "colapsa" a partir desse $i$, isto é

$$\forall j \geq i \; \Sigma_j^p = \Sigma_i^p$$

**A questão**

$$\boxed{\mathbf{P \neq NP?}}$$

**é uma das questões mais importantes – talvez a questão mais importante – em aberto na Ciência de Computadores.**

# Algoritmos aleatorizados

7 é um número aleatório? (*Don Knuth)*

Dados: matrizes $A$, $B$ e $C$ (num qualquer corpo $(K, +, \times)$) todas $n \times n$, verifica-se que $AB = C$?

O método directo (modelo uniforme) é $O(n^3)$
– ou $O(n^{2.373})$ usando o melhor algoritmo conhecido (muito complexo!).
Algoritmo que pode falhar mas é $O(n^2)$ e simples:

1. Escolhe um vector aleatório $r \in \{0,1\}^n$ de forma uniforme
2. Calcula $Br$ e depois $x = A(Br)$ (tempo $O(n^2)$)
3. Calcula $y = Cr$ (tempo $O(n^2)$)
4. Se $x = y$ responde SIM
5. Se $x \neq y$ responde NÃO

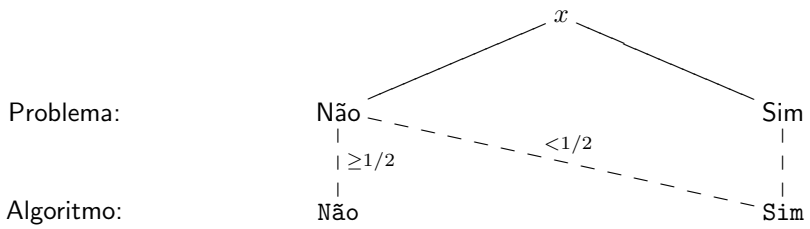
Funciona em qualquer corpo $(A, +, \times)$.

Seja qual for a instância $A, B, C$:

- Se a solução é SIM a resposta é SIM; Se a solução é NÃO a resposta é NÃO com probabilidade $\geq 1/2$
- A probabilidade de erro pode ser reduzida arbitrariamente (por repetição) mantendo o tempo $O(n^2)$

A aleatoridade não está na instância mas no algoritmo. Não se trata de uma análise do comportamento médio...

Este problema é P mas serve de base às ideias seguintes...




---

A resposta "`Sim`" do algoritmo é ambígua —
não-determinismo quando a resposta certa é "Não".

Há problemas para os quais só são conhecidos algoritmos polinomiais se se permite aleatorização e erro.

- Classe $RP$ (Randomized polynomial) – erro unilateral: linguagens $L$ que têm um algoritmo $A$ aleatorizado e polinomial (pior caso) tal que
  - Se $x \in L \Rightarrow prob(A \text{ aceitar}) \geq 1/2$
  - Se $x \notin L \Rightarrow prob(A \text{ aceitar}) = 0$
- Classe $BPP$ (Bounded error Probabilistic Polynomial) – erro bilateral: linguagens $L$ que têm algoritmo $A$ aleatorizado polinomial (pior caso) tal que
  - Se $x \in L \Rightarrow prob(A \text{ aceitar}) \geq 3/4$
  - Se $x \notin L \Rightarrow prob(A \text{ aceitar}) \leq 1/4$

Sabe-se por exemplo que

P$\subseteq$RP$\subseteq$NP (não se sabe se as inclusões são estritas)

(RP$\cup$co-RP) $\subseteq$ BPP (não se sabe se a inclusão é estrita)

Relação entre as classes

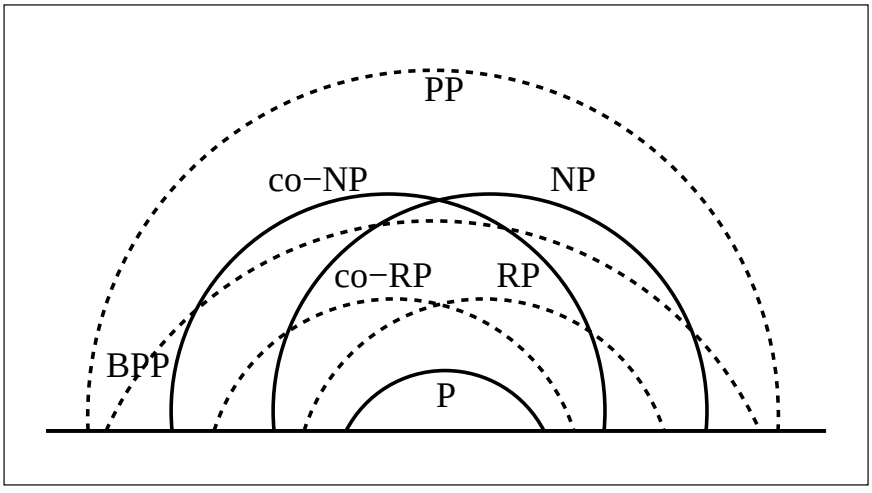
– RP, "Randomized polynomial": Aceita por maioria, rejeita sempre

– co-RP: Complementar de RP

– BPP: "Bounded error Probabilistic Polynomial"

– PP: "Probabilistic Polynomial": Máquina não determinística que aceita sse $> 1/2$ das possíveis computações aceitam.

$$co - PP = PP$$



$$\mathsf{P} \subseteq \mathsf{RP} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$
$$\mathsf{RP} \subseteq \mathsf{BPP} \subseteq PP$$
$$\mathsf{PP} = \text{co–PP}$$
$$\mathsf{BPP} = \text{co–BPP}$$
$$\mathsf{NP} \subseteq \mathsf{BPP} \subseteq \mathsf{PSPACE}$$
$$\mathsf{NP} \subseteq \mathsf{BPP} \Rightarrow \mathsf{NP} = \mathsf{RP}$$

Não se sabe se as inclusões são estritas...

$$\mathsf{R} = \text{co–RP?}$$
$$\mathsf{RP} \subseteq (\mathsf{NP} \cap \text{co-NP})?$$
$$\mathsf{BPP} \subseteq \mathsf{NP}$$

---

Há muitos exemplos de problemas em que o uso de algoritmos aleatorizados permite algoritmos mais *eficientes* e *simples*. Ver, por exemplo, o livro

Motwani and Raghavan: "Randomized Algorithms", Cambridge University Press, 1995.
e os trabalhos de Richard Karp.

## IP: ALEATÓRIOS+INTERACÇAO

Se permitirmos o uso de aleatórios e interacção (*teoria com conotações darwinistas?*) parece ser possível verificar (ter testemunhos) de problemas que não estão em NP (por exemplo, em co-NP\NP) com $O(1)$ bits!

---

**Exemplo**: Grafos não isomorfos, $GNI$
Nota: sabe-se que $GNI$ é $co-NP$, mas não se sabe se é P, $completoemNP$ ou intermédio.
Algoritmo
$V$: verificador polinomial com acesso a "bits" aleatórios.
P: Provador (=demonstrador) todo-poderoso que pode, em particular, conhecer $y$ mas não os aleatórios gerados por $V$.
Dados ($x$): dois grafos $G_1$ e $G_2$.
Pergunta: $G_1$ e $G_2$ não são isomorfos?

1. $V$ escolhe aleatóriamente um dos grafos $G_i$ e gera aleatoriamente (e uniformemente) uma sua cópia $H$ isomorfa (por permutação aleatória dos vértices).
   Envia $H$ a P, pedindo para este lhe responder com $j \in \{1,2\}$ tal que $H$ seja isomorfo a $G_j$.
2. P envia um $j \in \{1,2\}$ a $V$
3. $V$ responde da seguinte forma: se $i = j$ responde SIM (grafos não isomorfos); se $i \neq j$ responde NÃO (grafos isomorfos).

O algoritmo pode errar mas

- Se os grafos não são isomorfos existe um P que faz $V$ aceitar (P determina $j$ tal que $G_j$ seja isomorfo a $G_i$ – só há um).
- Se os grafos são isomorfos, qualquer que seja P, a probabilidade de $V$ rejeitar é $1/2$.

– *Só com 1 bit de informação correcto* (em vez de toda a informação contida em $y$), $V$ que é polinomial aceita de certeza um par de grafos não isomorfos.
– *Só com 1 bit de informação correcto ou falso*, $V$ rejeita $\geq 1/2$ dos pares de grafos isomorfos.
O erro poderia ser reduzido arbitrariamente...

---

A aleatorização e interacção parece permitir verificações com pouca informação mesmo em problemas que não pertencem a NP.
Sistema de demonstração interactiva de uma linguagem $L$:
Verificador polinomial e aleatorizado $V$ e demonstrador P em interacção, recebendo ambos os dados $x$ e satisfazendo

1. Completitude: se $x \in L$ o verificador aceita $x$.
2. Consistência: se $x \notin L$ e, seja qual for o demonstrador P, o verificador rejeita $x$ com probabilidade $\geq 1/2$.

Nota: Não há nada de especial no número $1/2$ (podia ser qualquer número $< 1$; repetindo um número fixo de vezes conseguir-se-ia qualquer $\varepsilon > 0$).
$IP(m())$: classe de linguagens para as quais existe um sistema de demonstração interactiva em que o número de mensagens trocadas entre P e $V$ não excede $m(|x|)$.
$IP$: número de mensagens limitado por um polinómio em $|x|$ ($IP = IP(poly)$).

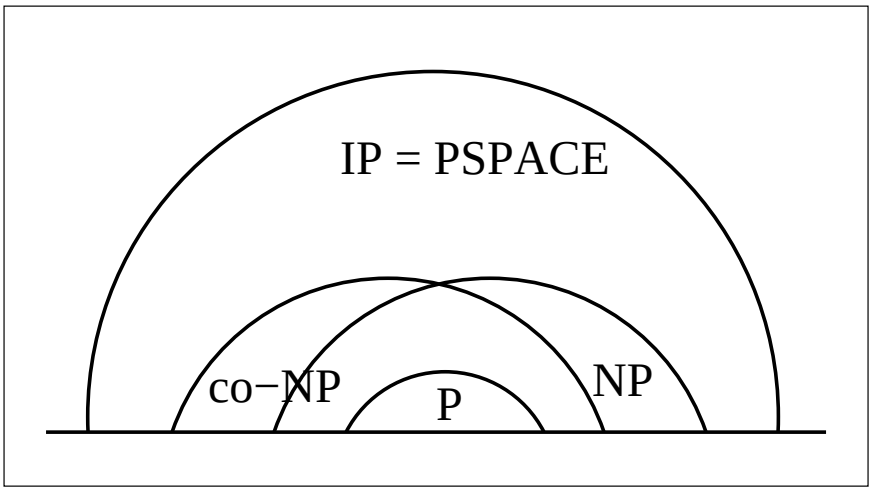
Notas:

- O uso da aleatoridade em $V$ é essencial. Não existindo, IP colapsa no sistema de demonstração NP.
- Porque é que $V$ não pergunta simplesmente a P se o problema tem solução? (Qualquer classe decidível seria $IP$!)

---

Vimos que $GNI \in IP$ e pode demonstrar-se que $GI \in IP$. Mais geralmente pode demonstrar-se que

$$\begin{aligned} NP &\subseteq IP \\ co-NP &\subseteq IP \end{aligned}$$



Alguns resultados

1. $L_1 \propto L_2,\ L_2 \in IP \ \Rightarrow\ L_1 \in IP$
   $\#3SAT$:
   **Instância.** $(U, C, s \in \mathbf{N})$
   **Pergunta.** : O conjunto de cláusulas pode ser satisfeito com (exactamente) $s$ atribuições de valores lógicos às variáveis $U$.
   Pode mostrar-se que $\#3SAT \in IP$.
   A demonstração utiliza uma técnica de "aritmetização" de um conjunto de cláusulas.

2. Para $s = 0$ temos o problema $INSAT$, insatisfabilidade de cláusulas.
   Como $INSAT$ é co-NP-completo resulta que $co - NP \subseteq IP$.
   Um caso particular que vimos é $GNI \in IP$

3. Pode mostrar-se que $3SAT \in IP$ e, por isso, NP⊆IP.

4. Também se pode mostrar directamente que NP⊆IP.

5. Pode mostrar-se que $IP = PSPACE$.

# Pequena bibliografia

[1] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.
Em especial: capítulos 2 e 3 e secções 7.1 e 7.2.

[2] Bernard M. Moret, *The Theory of Computation*, Addison-Wesley, 1997.
Nota. Inclui também a teoria da computabilidade.

[3] Michael Spiser, *Introduction to the Theory of Computation*, PWS Publishing Company, 1997.